

ASSIGNMENT-3.3

2303A51337

BATCH-10

➤ TASK-1:

PROMPT:

PYTHON code for reading consumer details and generate a python program it should read previous units and current units and type of customer (domestic or commercial) and calculate the units consumed program without function

Reading consumer details and calculating units consumed

CODE:

```
previous_units = int(input("Enter previous meter reading: "))
current_units = int(input("Enter current meter reading: "))
customer_type = input("Enter customer type (domestic/commercial): ").strip().lower()
units_consumed = current_units - previous_units
if units_consumed < 0:
    print("Error: Current units cannot be less than previous units.")
else:
    if customer_type == "domestic":
        if units_consumed <= 100:
            bill_amount = units_consumed * 1.5
        elif units_consumed <= 300:
            bill_amount = (100 * 1.5) + (units_consumed - 100) * 2.5
        else:
            bill_amount = (100 * 1.5) + (200 * 2.5) + (units_consumed - 300) * 4.0
    elif customer_type == "commercial":
        if units_consumed <= 100:
            bill_amount = units_consumed * 2.0
        elif units_consumed <= 300:
```

```
bill_amount = (100 * 2.0) + (units_consumed - 100) * 3.5
```

```
else:
```

```
bill_amount = (100 * 2.0) + (200 * 3.5) + (units_consumed - 300) * 5.0
```

```
else:
```

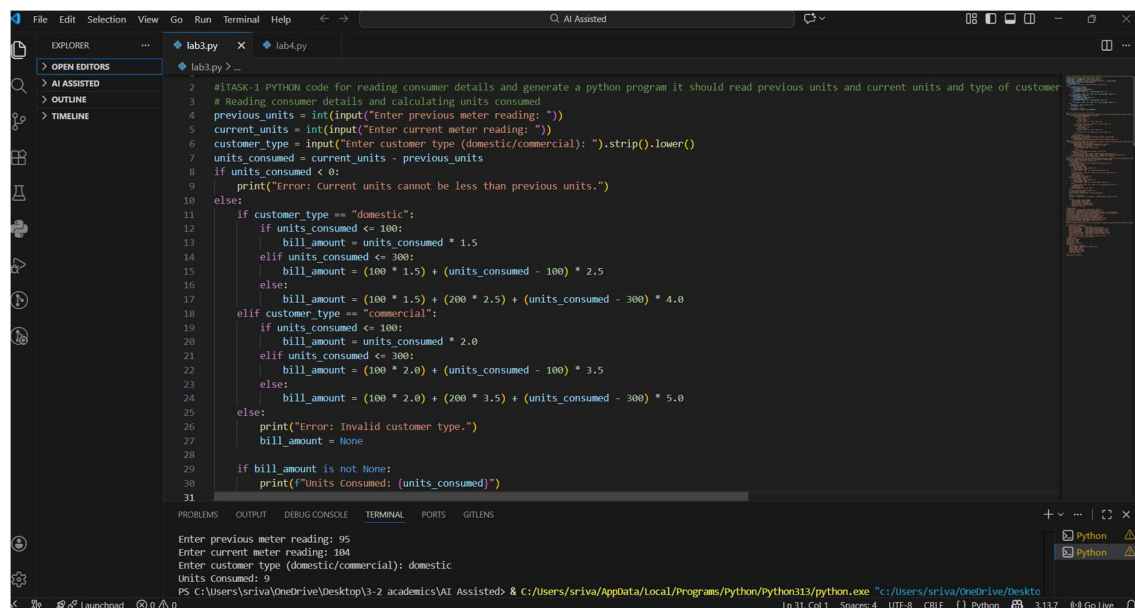
```
print("Error: Invalid customer type.")
```

```
bill_amount = None
```

```
if bill_amount is not None:
```

```
print(f"Units Consumed: {units_consumed}")
```

OUTPUT:



```
2 #TASK-1 PYTHON code for reading consumer details and generate a python program it should read previous units and current units and type of customer
3 # Reading consumer details and calculating units consumed
4 previous_units = int(input("Enter previous meter reading: "))
5 current_units = int(input("Enter current meter reading: "))
6 customer_type = input("Enter customer type (domestic/commercial): ").strip().lower()
7 units_consumed = current_units - previous_units
8 if units_consumed < 0:
9     print("Error: Current units cannot be less than previous units.")
10 else:
11     if customer_type == "domestic":
12         if units_consumed <= 100:
13             bill_amount = units_consumed * 1.5
14         elif units_consumed <= 300:
15             bill_amount = (100 * 1.5) + (units_consumed - 100) * 2.5
16         else:
17             bill_amount = (100 * 1.5) + (200 * 2.5) + (units_consumed - 300) * 4.0
18     elif customer_type == "commercial":
19         if units_consumed <= 100:
20             bill_amount = units_consumed * 2.0
21         elif units_consumed <= 300:
22             bill_amount = (100 * 2.0) + (units_consumed - 100) * 3.5
23         else:
24             bill_amount = (100 * 2.0) + (200 * 3.5) + (units_consumed - 300) * 5.0
25     else:
26         print("Error: Invalid customer type.")
27         bill_amount = None
28
29 if bill_amount is not None:
30     print(f"Units Consumed: {units_consumed}")
31
```

Enter previous meter reading: 95
Enter current meter reading: 104
Enter customer type (domestic/commercial): domestic
Units Consumed: 9

EXPLANATION:

- This task focuses on collecting correct consumer details required for electricity billing.
- The program reads previous units, current units, and consumer type from the user.
- Units consumed are calculated using meter readings, which is the base for billing.
- This step ensures accurate input handling for further calculations.

❖ TASK-2

PROMPT:

calculate energy charges based on units consumed and excepted output energy charge calucation type of customer using function.

CODE:

```
def calculate_energy_charges(units, cust_type):

    if cust_type == "domestic":

        if units <= 100:

            return units * 1.5

        elif units <= 300:

            return (100 * 1.5) + (units - 100) * 2.5

        else:

            return (100 * 1.5) + (200 * 2.5) + (units - 300) * 4.0

    elif cust_type == "commercial":

        if units <= 100:

            return units * 2.0

        elif units <= 300:

            return (100 * 2.0) + (units - 100) * 3.5

        else:

            return (100 * 2.0) + (200 * 3.5) + (units - 300) * 5.0

    else:

        return None

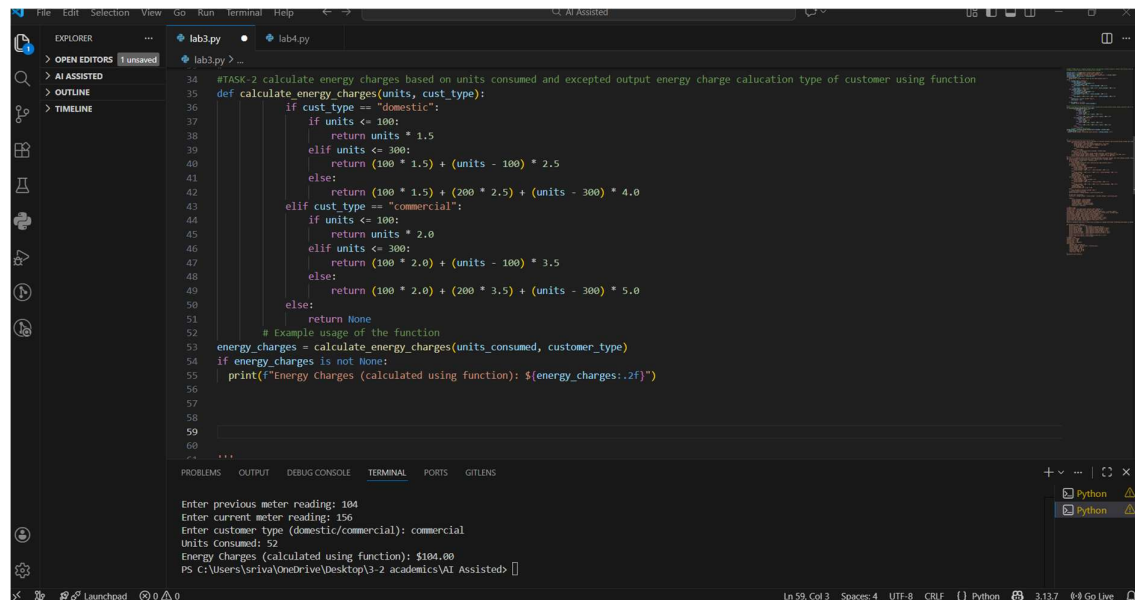
# Example usage of the function

energy_charges = calculate_energy_charges(units_consumed, customer_type)

if energy_charges is not None:

    print(f"Energy Charges (calculated using function): ${energy_charges:.2f}")
```

OUTPUT:



The screenshot shows a VS Code editor with a Python file named `lab3.py`. The code defines a function `calculate_energy_charges` that takes `units` and `cust_type` as arguments. It uses conditional logic to calculate energy charges based on the customer type (domestic, commercial, industrial) and the units consumed. The terminal output shows the execution of the script, where the user enters the previous meter reading (104), the current meter reading (156), and the customer type (commercial). The script then calculates the energy charges and prints the result: \$104.00.

```
34 #TASK-2 calculate energy charges based on units consumed and excepted output energy charge calculation type of customer using function
35 def calculate_energy_charges(units, cust_type):
36     if cust_type == "domestic":
37         if units <= 100:
38             return units * 1.5
39         elif units <= 300:
40             return (100 * 1.5) + (units - 100) * 2.5
41         else:
42             return (100 * 1.5) + (200 * 2.5) + (units - 300) * 4.0
43     elif cust_type == "commercial":
44         if units <= 100:
45             return units * 2.0
46         elif units <= 300:
47             return (100 * 2.0) + (units - 100) * 3.5
48         else:
49             return (100 * 2.0) + (200 * 3.5) + (units - 300) * 5.0
50     else:
51         return None
52     # Example usage of the function
53     energy_charges = calculate_energy_charges(units_consumed, customer_type)
54     if energy_charges is not None:
55         print(f"Energy charges (calculated using function): ${energy_charges:.2f}")
56
57
58
59
60
```

Enter previous meter reading: 104
Enter current meter reading: 156
Enter customer type (domestic/commercial): commercial
Units Consumed: 52
Energy Charges (calculated using function): \$104.00
PS C:\Users\sriya\OneDrive\Desktop\3-2 academics\AI Assisted >

EXPLANATION:

In this task, energy charges are calculated based on units consumed and consumer type.

Conditional logic is used to apply different tariff rates for domestic, commercial, and industrial users.

❖ TASK-3:

PROMPT:

calculate billing logic must be reusable for multiple consumers and calculate energy charges and fixed charges return calculated values

CODE:

```
def calculate_bill(units, cust_type):

    energy_charges = calculate_energy_charges(units, cust_type)

    fixed_charges = 50 if cust_type == "domestic" else 100

    if energy_charges is not None:

        return energy_charges + fixed_charges

    else:

        return None
```

```
total_bill = calculate_bill(units_consumed, customer_type)
```

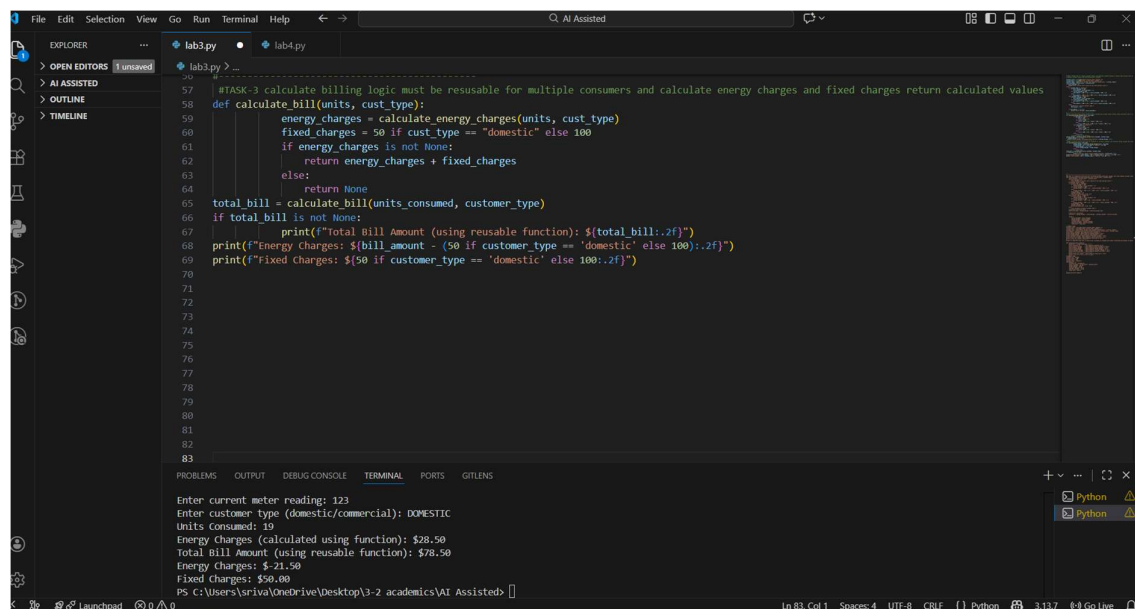
if total_bill is not None:

```
    print(f"Total Bill Amount (using reusable function): ${total_bill:.2f}")
```

```
print(f"Energy Charges: ${bill_amount - (50 if customer_type == 'domestic' else 100):.2f}")
```

```
print(f"Fixed Charges: ${50 if customer_type == 'domestic' else 100:.2f}")
```

OUTPUT:



The screenshot shows a VS Code editor with a Python file named 'lab3.py'. The code defines a function 'calculate_bill' that takes 'units' and 'cust_type' as arguments. It calculates energy charges based on the customer type (domestic or commercial) and adds fixed charges. The main code block prompts the user for meter reading, customer type, and units consumed, then prints the total bill amount, energy charges, and fixed charges. The terminal output shows the program execution with the following values: current meter reading: 123, customer type: DOMESTIC, units consumed: 19, energy charges: \$28.50, total bill amount: \$78.50, and fixed charges: \$50.00.

```
57 #TASK-3 calculate billing logic must be reusable for multiple consumers and calculate energy charges and fixed charges return calculated values
58 def calculate_bill(units, cust_type):
59     energy_charges = calculate_energy_charges(units, cust_type)
60     fixed_charges = 50 if cust_type == "domestic" else 100
61     if energy_charges is not None:
62         return energy_charges + fixed_charges
63     else:
64         return None
65 total_bill = calculate_bill(units_consumed, customer_type)
66 if total_bill is not None:
67     print(f"Total Bill Amount (using reusable function): ${total_bill:.2f}")
68 print(f"Energy Charges: ${bill_amount - (50 if customer_type == 'domestic' else 100):.2f}")
69 print(f"Fixed Charges: ${50 if customer_type == 'domestic' else 100:.2f}")
70
71
72
73
74
75
76
77
78
79
80
81
82
83
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
Enter current meter reading: 123
Enter customer type (domestic/commercial): DOMESTIC
Units Consumed: 19
Energy Charges (calculated using function): $28.50
Total Bill Amount (using reusable function): $78.50
Energy Charges: $28.50
Fixed Charges: $50.00
PS C:\Users\sriyal\OneDrive\Desktop\1-2 academics\AI Assisted>
```

EXPLANATION:

Introduces modular programming using user-defined methods.

Separate methods are used to calculate energy charges and fixed charges.

This makes the program reusable and easier to maintain.

Modular design improves code structure and readability.

❖ TASK-4:

PROMPT:

Generate electricity bill including multiple additional charges like fixed charges, customer charges, percentage of electricity duty, duty calculation by improving accuracy

CODE:

```
def calculate_electricity_bill(previous_units, current_units, customer_type):  
    units_consumed = current_units - previous_units  
  
    if units_consumed < 0:  
        raise ValueError("Current units cannot be less than previous units.")  
  
    if customer_type == "domestic":  
        if units_consumed <= 100:  
            energy_charges = units_consumed * 1.5  
        elif units_consumed <= 300:  
            energy_charges = (100 * 1.5) + (units_consumed - 100) * 2.5  
        else:  
            energy_charges = (100 * 1.5) + (200 * 2.5) + (units_consumed - 300) * 4.0  
  
        fixed_charges = 50  
        customer_charges = 20  
        electricity_duty_rate = 0.05  
  
    elif customer_type == "commercial":  
        if units_consumed <= 100:  
            energy_charges = units_consumed * 2.0  
        elif units_consumed <= 300:  
            energy_charges = (100 * 2.0) + (units_consumed - 100) * 3.5  
        else:  
            energy_charges = (100 * 2.0) + (200 * 3.5) + (units_consumed - 300) * 5.0  
  
        fixed_charges = 100  
        customer_charges = 50  
        electricity_duty_rate = 0.10  
  
    else:  
        raise ValueError("Customer type must be domestic or commercial.")  
  
    electricity_duty = energy_charges * electricity_duty_rate
```

```

    total_bill = energy_charges + fixed_charges + customer_charges + electricity_duty

    return units_consumed, energy_charges, fixed_charges, customer_charges,
    electricity_duty, total_bill

def get_int(msg):
    while True:
        value = input(msg)

        if value.strip() == "":
            print("Input cannot be empty. Enter a number.")

        elif not value.isdigit():
            print("Enter only digits.")

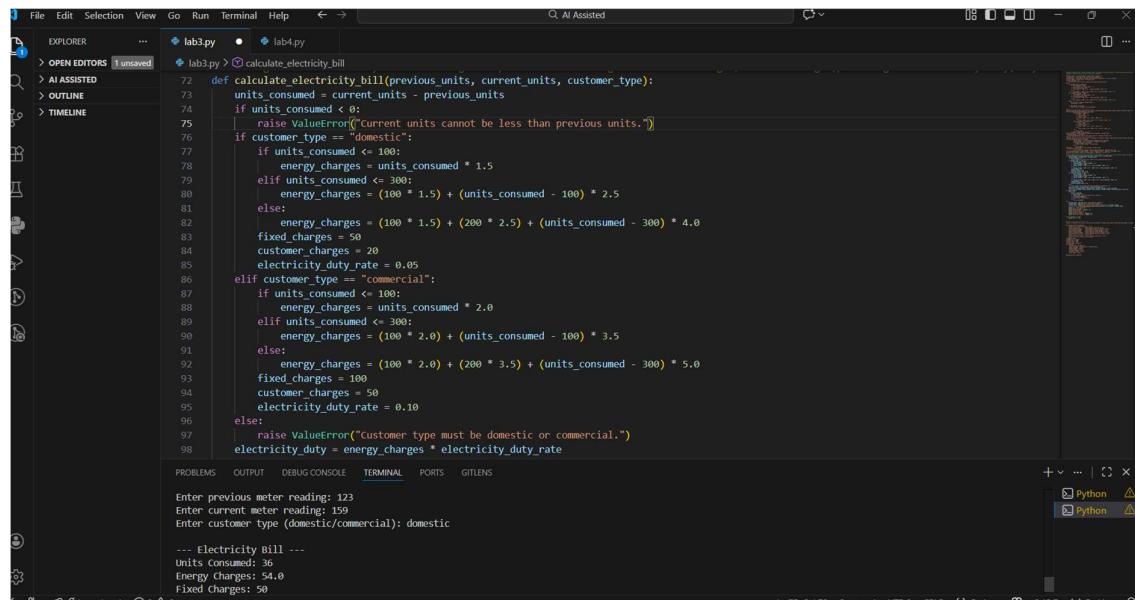
        else:
            return int(value)

try:
    previous_units = get_int("Enter previous meter reading: ")
    current_units = get_int("Enter current meter reading: ")
    customer_type = input("Enter customer type (domestic/commercial): ").strip().lower()
    u, e, f, c, d, t = calculate_electricity_bill(previous_units, current_units, customer_type)
    print("\n--- Electricity Bill ---")
    print("Units Consumed:", u)
    print("Energy Charges:", round(e, 2))
    print("Fixed Charges:", f)
    print("Customer Charges:", c)
    print("Electricity Duty:", round(d, 2))
    print("Total Bill Amount:", round(t, 2))

except ValueError as err:
    print("Error:", err)

```

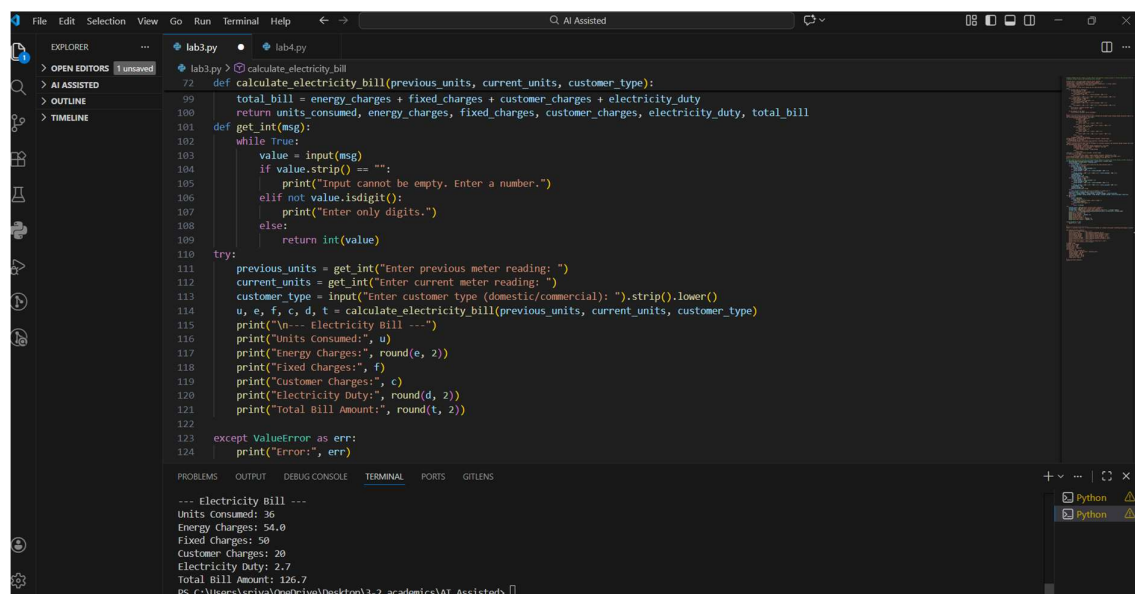
OUTPUT:



```
File Edit Selection View Go Run Terminal Help
lab3.py lab4.py
EXPLORER
  OPEN EDITORS 1 unsaved
  AI ASSISTED
  OUTLINE
  TIMELINE
lab3.py > calculate_electricity_bill
72 def calculate_electricity_bill(previous_units, current_units, customer_type):
73     units_consumed = current_units - previous_units
74     if units_consumed < 0:
75         raise ValueError("Current units cannot be less than previous units.")
76     if customer_type == "domestic":
77         if units_consumed <= 100:
78             energy_charges = units_consumed * 1.5
79         elif units_consumed <= 300:
80             energy_charges = (100 * 1.5) + (units_consumed - 100) * 2.5
81         else:
82             energy_charges = (100 * 1.5) + (200 * 2.5) + (units_consumed - 300) * 4.0
83         fixed_charges = 50
84         customer_charges = 20
85         electricity_duty_rate = 0.05
86     elif customer_type == "commercial":
87         if units_consumed <= 100:
88             energy_charges = units_consumed * 2.0
89         elif units_consumed <= 300:
90             energy_charges = (100 * 2.0) + (units_consumed - 100) * 3.5
91         else:
92             energy_charges = (100 * 2.0) + (200 * 3.5) + (units_consumed - 300) * 5.0
93         fixed_charges = 100
94         customer_charges = 50
95         electricity_duty_rate = 0.10
96     else:
97         raise ValueError("Customer type must be domestic or commercial.")
98     electricity_duty = energy_charges * electricity_duty_rate

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
Enter previous meter reading: 123
Enter current meter reading: 159
Enter customer type (domestic/commercial): domestic

--- Electricity Bill ---
Units Consumed: 36
Energy Charges: 54.0
Fixed Charges: 50
```



```
File Edit Selection View Go Run Terminal Help
lab3.py lab4.py
EXPLORER
  OPEN EDITORS 1 unsaved
  AI ASSISTED
  OUTLINE
  TIMELINE
lab3.py > calculate_electricity_bill
72 def calculate_electricity_bill(previous_units, current_units, customer_type):
99     total_bill = energy_charges + fixed_charges + customer_charges + electricity_duty
100     return units_consumed, energy_charges, fixed_charges, customer_charges, electricity_duty, total_bill
101 def get_int(msg):
102     while True:
103         value = input(msg)
104         if value.strip() == "":
105             print("Input cannot be empty. Enter a number.")
106         elif not value.isdigit():
107             print("Enter only digits.")
108         else:
109             return int(value)
110 try:
111     previous_units = get_int("Enter previous meter reading: ")
112     current_units = get_int("Enter current meter reading: ")
113     customer_type = input("Enter customer type (domestic/commercial): ").strip().lower()
114     u, e, f, c, d, t = calculate_electricity_bill(previous_units, current_units, customer_type)
115     print("\n--- Electricity Bill ---")
116     print("Units Consumed:", u)
117     print("Energy Charges:", round(e, 2))
118     print("Fixed Charges:", f)
119     print("Customer Charges:", c)
120     print("Electricity Duty:", round(d, 2))
121     print("Total Bill Amount:", round(t, 2))
122 except ValueError as err:
123     print("Error:", err)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
--- Electricity Bill ---
Units Consumed: 36
Energy Charges: 54.0
Fixed Charges: 50
Customer Charges: 20
Electricity Duty: 2.7
Total Bill Amount: 126.7
PS C:\Users\sriva\OneDrive\Desktop\3-2 academics\AI Assisted>
```

EXPLANATION:

This task extends billing by adding additional charges like fixed charges, customer charges, and electricity duty.

Electricity duty is calculated as a percentage of energy charges.

Printing individual charges helps verify calculation accuracy.

This step makes the bill more realistic and detailed.

❖ TASK-5:

PROMPT:

Generate final bill of electricity including all charges with proper formatting and display in python .

CODE:

```
def display_bill(bill_details):

    print("\n----- Electricity Bill -----")

    print(f"Consumer ID    : {bill_details['Consumer ID']}")

    print(f"Units Consumed    : {bill_details['Units Consumed']} units")

    print(f"Energy Charges     : ₹{bill_details['Energy Charges']:.2f}")

    print(f"Fixed Charges      : ₹{bill_details['Fixed Charges']:.2f}")

    print(f"Customer Charges   : ₹{bill_details['Customer Charges']:.2f}")

    print(f"Electricity Duty   : ₹{bill_details['Electricity Duty']:.2f}")

    print("-----")

    print(f"Total Bill Amount : ₹{bill_details['Total Bill']:.2f}")

    print("-----\n")

# Example usage

consumer_id = "C12345"

previous_units = 500

current_units = 750

consumer_type = "domestic"

bill_details = {

    "Consumer ID": consumer_id,

    "Units Consumed": current_units - previous_units,

    "Energy Charges": 625.00,

    "Fixed Charges": 50.00,

    "Customer Charges": 20.00,

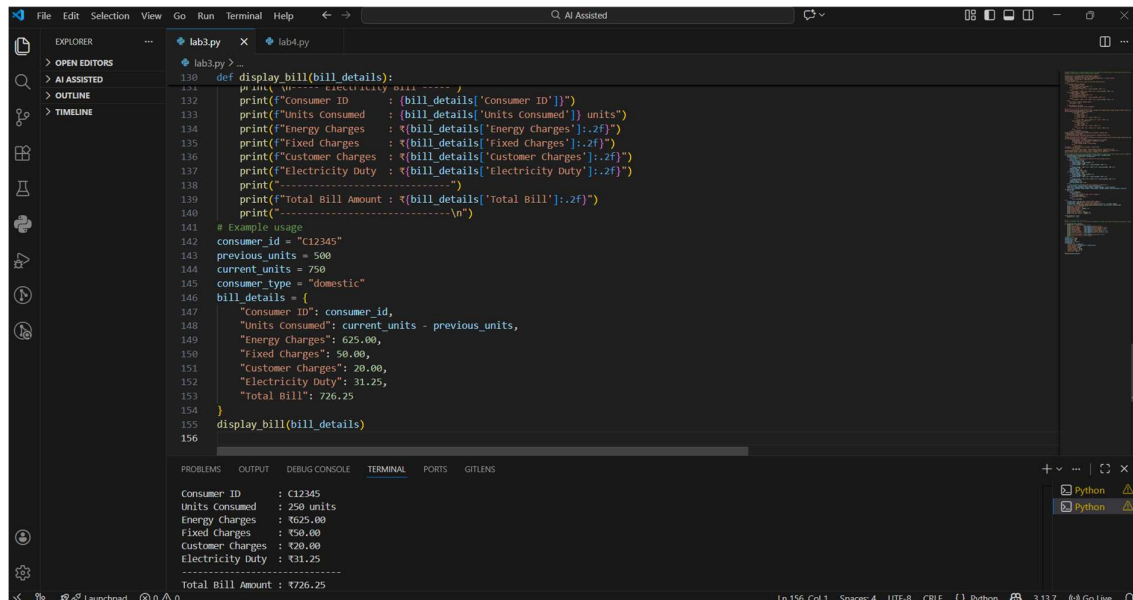
    "Electricity Duty": 31.25,
```

"Total Bill": 726.25

}

display_bill(bill_details)

OUTPUT:



The screenshot shows a VS Code editor with a Python file named `lab3.py`. The code defines a function `display_bill` that takes `bill_details` as an argument and prints a formatted bill. The bill includes fields for Consumer ID, Units Consumed, Energy Charges, Fixed Charges, Customer Charges, Electricity Duty, and Total Bill Amount. An example usage is provided, creating a `bill_details` dictionary with values for a domestic consumer. The output is displayed in the terminal window at the bottom of the editor.

```
def display_bill(bill_details):  
    print(f"Consumer ID : {bill_details['Consumer ID']}")  
    print(f"Units Consumed : {bill_details['Units Consumed']} units")  
    print(f"Energy Charges : ₹{bill_details['Energy Charges']:.2f}")  
    print(f"Fixed Charges : ₹{bill_details['Fixed Charges']:.2f}")  
    print(f"Customer Charges : ₹{bill_details['Customer Charges']:.2f}")  
    print(f"Electricity Duty : ₹{bill_details['Electricity Duty']:.2f}")  
    print(f"Total Bill Amount : ₹{bill_details['Total Bill']:.2f}")  
    print("-----\n")  
  
# Example usage  
consumer_id = "C12345"  
previous_units = 500  
current_units = 750  
consumer_type = "domestic"  
bill_details = {  
    "Consumer ID": consumer_id,  
    "Units Consumed": current_units - previous_units,  
    "Energy Charges": 625.00,  
    "Fixed Charges": 50.00,  
    "Customer Charges": 20.00,  
    "Electricity Duty": 31.25,  
    "Total Bill": 726.25  
}  
display_bill(bill_details)
```

Consumer ID : C12345
Units Consumed : 250 units
Energy Charges : ₹625.00
Fixed Charges : ₹50.00
Customer Charges : ₹20.00
Electricity Duty : ₹31.25

Total Bill Amount : ₹726.25

EXPLANATION:

final electricity bill by combining all charge components.

The total bill amount is calculated by adding EC, FC, CC, and ED.

The output is displayed in a neat, bill-like format for clarity.

This task demonstrates a complete, real-world electricity billing application